

Single Image Haze Removal Using Dark Channel Prior

22 апреля 2019 г.

```
In [1]: from __future__ import division

import logging

import cv2
import numpy as np
from numpy.lib.stride_tricks import as_strided
import scipy.sparse
import scipy.sparse.linalg
import numpy as np
import cv2 as cv
import matplotlib.pyplot as plt
import scipy.ndimage as ndimage

In [2]: def bgr2rgb(img):
    b,g,r = cv.split(img)
    return cv.merge([r,g,b])

def rgb2bgr(img):
    b,g,r = cv.split(img)
    return cv.merge([r,g,b])

In [3]: img = cv.imread("images/hongkong_foggy.jpg")
plt.imshow(img)
plt.axis("off");
```

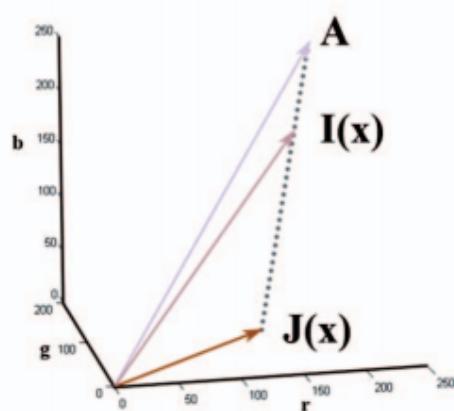


$$\mathbf{I}(\mathbf{x}) = \mathbf{J}(\mathbf{x})t(\mathbf{x}) + \mathbf{A}(1 - t(\mathbf{x})) \quad (1)$$

где \mathbf{I} это наблюдаемая интенсивность, \mathbf{J} светимость объектов, \mathbf{A} общее свечение атмосферы, t средняя передача описываемая порцией света, которая не рассеивается и достигает камеры.

$$t(\mathbf{x}) = \frac{\|\mathbf{A} - \mathbf{I}(\mathbf{x})\|}{\|\mathbf{A} - \mathbf{J}(\mathbf{x})\|} = \frac{A^c - I^c(\mathbf{x})}{A^c - J^c(\mathbf{x})} \quad (2)$$

где $c \in \{r, g, b\}$ индекс цвета.



Идея Dark Channel Prior

Метод основывается на следующем наблюдении: в большинстве участков незатуманенных картинок без неба, по крайней мере, один цветной канал имеет несколько пикселей,

интенсивность которых очень низкая и близка к нулю. Следовательно, минимальная интенсивность в таком участке близка к нулю.

$$J^{dark} = \min_{\mathbf{y} \in \Omega(\mathbf{x})} \left(\min_{c \in \{r, g, b\}} J^c(\mathbf{y}) \right) \quad (3)$$

$$J^{dark} \rightarrow 0$$

```
In [4]: def get_radiance(image, r):
    r2 = int(int(r)/2)
    w, h, d = image.shape
    padded = np.pad(image, ((r2, r2), (r2, r2), (0, 0)), 'edge')
    radiance = np.zeros((w, h))
    for i, j in np.ndindex(radiance.shape):
        radiance[i, j] = np.min(padded[i:i + r, j:j + r, :])
    return radiance

In [5]: rad = get_radiance(img, r = 15)
plt.imshow(rad, cmap = 'gray')
plt.suptitle("radiance")
plt.axis('off')
```

radiance



Оценка передачи $t(\mathbf{x})$

Будем работать в предположении, что \mathbf{A} уже известно. Перепишем (1) в виде:

$$\frac{I^c(x)}{A^c} = t(x) \frac{J^c(x)}{A^c} + 1 - t(x) \quad (4)$$

Далее предполагаем, что передача внутри участка $\Omega(\mathbf{x})$ является постоянной. Обозначим эту передачу как \tilde{t} . Затем, рассчитаем минимальную интенсивность с обеих сторон (4).

$$\min_{\mathbf{y} \in \Omega(\mathbf{x})} \left(\min_c \frac{I^c(\mathbf{x})}{A^c} \right) = \tilde{t}(\mathbf{x}) \min_{\mathbf{y} \in \Omega(\mathbf{x})} \left(\min_c \frac{J^c(\mathbf{y})}{A^c} \right) + 1 - \tilde{t}(\mathbf{x}) \quad (5)$$

т.к. A^c положительно, значит можем переписать (3):

$$\frac{J^{dark}}{A^c} = \min_{\mathbf{y} \in \Omega(\mathbf{x})} \left(\min_{c \in \{r,g,b\}} \frac{J^c(\mathbf{y})}{A^c} \right) \rightarrow 0 \quad (6)$$

Подставляя (6) в (5), получаем:

$$\tilde{t}(\mathbf{x}) = 1 - \min_{\mathbf{y} \in \Omega(\mathbf{x})} \left(\min_{c \in \{r,g,b\}} \frac{I^c(\mathbf{y})}{A^c} \right) \quad (7)$$

Исследования показывают, что наличие дымки является сигналом для человека, по которому он осознает глубину. Это явление называется воздушной перспективой. Если мы удалим туман абсолютно, изображение может показаться неестественным, и мы можем потерять ощущение глубины. Так, можем выборочно сохранять очень небольшое количество тумана для далеких объектов с помощью введения постоянного параметра ω ($0 < \omega < 1$) в (7):

$$\tilde{t}(\mathbf{x}) = 1 - \omega \min_{\mathbf{y} \in \Omega(\mathbf{x})} \left(\min_c \frac{I^c(\mathbf{y})}{A^c} \right) \quad (8)$$

Мягкий Матирование

Заметим, что уравнение изображения дымки (1) имеет аналогичную форму, как уравнение матирования изображения:

$$\mathbf{I} = \mathbf{F}\alpha + \mathbf{B}(1 - \alpha) \quad (9)$$

где \mathbf{F} и \mathbf{B} цвета переднего и заднего планов соответственно, и α является непрозрачностью переднего плана. Карта передачи в уравнении изображения дымки — это альфа-карта. Поэтому, мы можем применить closed-form framework of matting для оценки передачи.

Обозначим уточненную карту передачи $t(\mathbf{x})$. Перезапишем $t(\mathbf{x})$ и $\tilde{t}(\mathbf{x})$ в виде векторов как \mathbf{t} и $\tilde{\mathbf{t}}$, и задача минимизировать следующую функцию:

$$E(\mathbf{t}) = \mathbf{t}^T L \mathbf{t} + \lambda (\mathbf{t} - \tilde{\mathbf{t}})^T (\mathbf{t} - \tilde{\mathbf{t}}) \quad (10)$$

L называется Лапласиан:

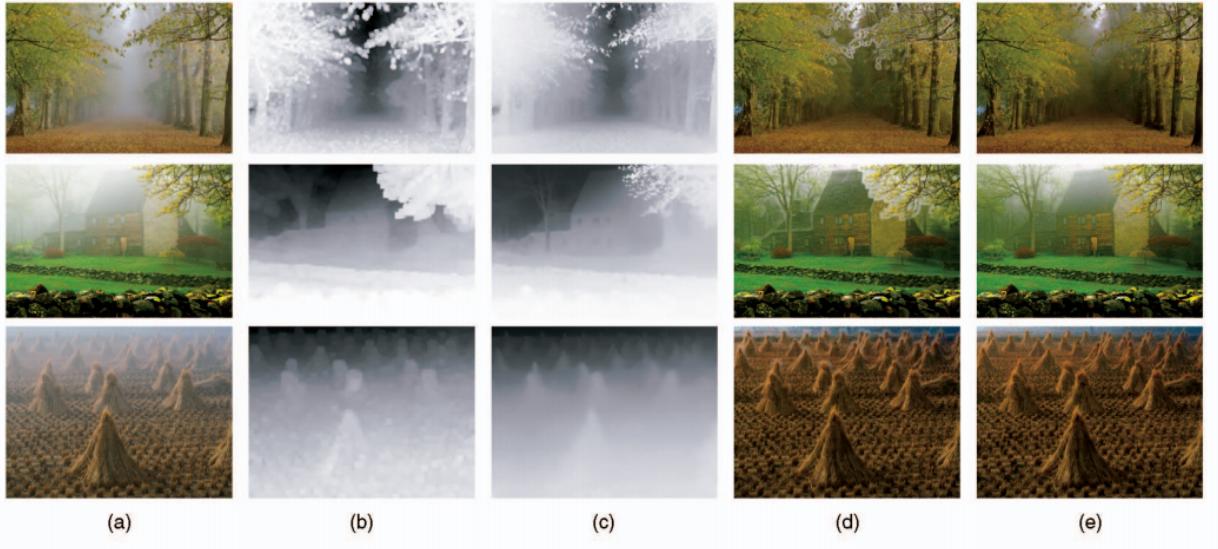
$$L_{ij} = \sum_{k|(i,j) \in w_k} \left(\delta_{ij} - \frac{1}{|w_k|} \left(1 + (\mathbf{I}_i - \mu_k)^T \left(\Sigma_k + \frac{\epsilon}{|w_k|} U_3 \right)^{-1} (\mathbf{I}_j - \mu_k) \right) \right) \quad (11)$$

где \mathbf{I}_i и \mathbf{I}_j — цвета изображения \mathbf{I} в пикселях i и j , δ_{ij} — символ Кронекера, μ_k и Σ_k -средняя и ковариационная матрицы цветов в окне w_k , U_3 представляет собой 3×3 единичная матрица, ϵ является регуляризующим параметром, а $|w_k|$ - количество пикселей в окне.

Оптимальная \mathbf{t} может быть получена путем решения следующей разреженной линейной системы:

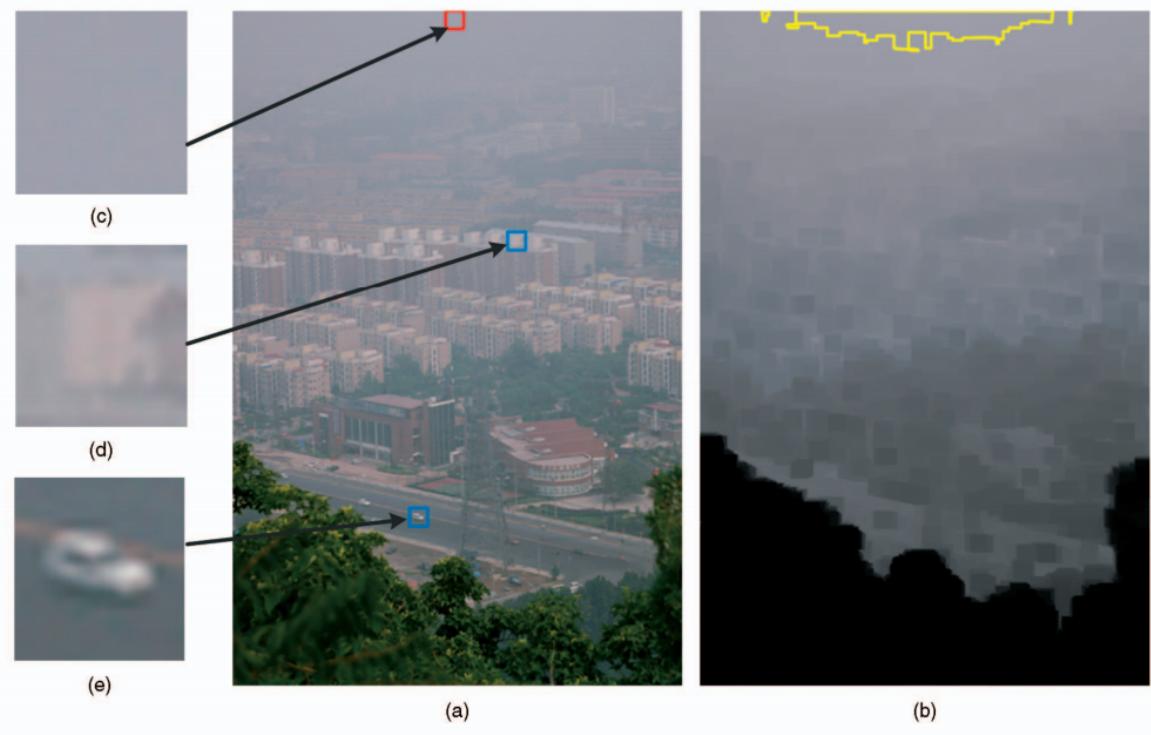
$$(L + \lambda U)\mathbf{t} = \lambda \tilde{\mathbf{t}} \quad (12)$$

После решения линейной системы (12) применим двусторонний фильтр “Bilateral Filtering for Gray and Color Images” на \mathbf{t} для сглаживания текстур малого масштаба. На рис.(c) показаны уточненные результаты, с помощью рис.(b) как ограничение. Как мы видим, нимбы и артефакты блоков скрыты.



Оценка общего светчения атмосферы

Когда пиксели на бесконечном расстоянии ($t \approx 0$), самое яркое \mathbf{I} является наиболее затуманенным и приблизительно равно A . Но на практике мы редко можем игнорировать солнечный свет.



В этой ситуации самый яркий пиксель всего изображения может быть ярче атмосферного света. Он может быть на белом автомобиле или в белом здании (Рис.(d) и (e)).

Как обсуждалось ранее, минимальная яркость затуманенного изображения приблизительно соответствует плотности тумана (см. Рис. (6)). Таким образом, мы можем использовать её, чтобы обнаружить наиболее непрозрачную область и улучшить оценку атмосферного свечения. Сначала мы выбираем самые яркие ($0,1\%$) пикселей минимальной яркости. Эти пиксели, как правило, наиболее затуманены (ограничены желтыми линиями на рис. (6)). Среди этих пикселей, выбираются пиксели с самой высокой интенсивностью во входном изображении в качестве атмосферного свечения. Эти пиксели находятся в красном прямоугольнике на рис.(a). Обратите внимание, что эти пиксели могут быть не самыми яркими во всем входном изображении.

Воссоздание изображения

С помощью атмосферного свечения и карты передачи мы можем восстановить изображение согласно (1). Но слагаемое прямого затухания $\mathbf{J}(\mathbf{x})t(\mathbf{x})$ может быть очень близко к нулю, когда $t(\mathbf{x})$ близка к нулю. Непосредственно восстановленное изображения подвержено шуму. Поэтому мы ограничиваем передачу, используя нижнюю границу t_0 , то есть мы сохраняем небольшое количество помутнения в очень плотных областях. В итоге получаем:

$$\mathbf{J}(\mathbf{x}) = \frac{\mathbf{I}(\mathbf{x}) - \mathbf{A}}{\max(t(\mathbf{x}), t_0)} + \mathbf{A} \quad (13)$$

Значение t_0 составляет 0,1. Поскольку сияние объектов обычно не такое яркое, как атмосферное свечение, изображение после удаления тумана выглядит тусклым. Таким образом, мы увеличиваем экспозицию $\mathbf{J}(\mathbf{x})$.

```
In [6]: def get_atmosphere_A(image, radiance, p = 0.001):
    w, h, _ = image.shape
    pixels = image.reshape(w * h, 3)
    radiance = radiance.reshape(w * h)
    indexes = (-radiance).argsort()[:int(w * h * p)]

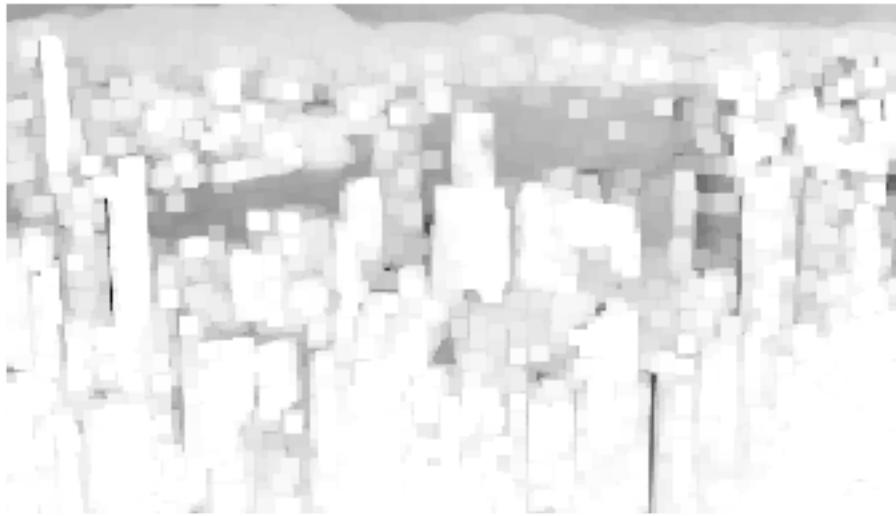
    return np.max(pixels.take(indexes, axis=0), axis=0)
```

```
In [7]: k = int(0.001*np.prod(img.shape[:2]))
idx = np.argpartition(-rad.ravel(),k)[:k]
x_y = np.column_stack(np.unravel_index(idx, rad.shape))
x, y = np.hsplit(x_y,2)
plt.imshow(img)
plt.scatter(y, x, c='r')
plt.axis('off')
plt.show();
```



```
In [8]: def get_t_waved(img, A, r = 15, omega = 0.95):
    norm_img = img / A
    norm_img_dc = get_radianc(norm_img, r = r)
    return 1 - omega * norm_img_dc
```

```
In [9]: A = get_atmosphere_A(img, rad)
alpha_map = get_t_waved(img, A)
plt.imshow(alpha_map, cmap='gray')
plt.axis("off");
```



```
In [10]: def _rolling_block(A, block=(3, 3)):
    shape = (A.shape[0] - block[0] + 1, A.shape[1] - block[1] + 1) + block
    strides = (A.strides[0], A.strides[1]) + A.strides
    return as_strided(A, shape=shape, strides=strides)

def compute_laplacian_my(img, mask=None, eps=10**(-7), win_rad=1):
    win_size = (win_rad * 2 + 1) ** 2
    h, w, d = img.shape
    c_h, c_w = h - 2 * win_rad, w - 2 * win_rad
    win_diam = win_rad * 2 + 1

    indsM = np.arange(h * w).reshape((h, w))
    ravelImg = img.reshape(h * w, d)
    indsM = np.pad(indsM, ((win_rad, win_rad), (win_rad, win_rad)), 'edge')

    win_inds = _rolling_block(indsM, block=(win_diam, win_diam))

    win_inds = win_inds.reshape(h, w, win_size)

    win_inds = win_inds.reshape(-1, win_size)
```

```

winI = ravelImg[win_inds]

win_mu = np.mean(winI, axis=1, keepdims=True)
win_var = np.einsum('...ji,...jk ->...ik', winI, winI) / win_size -
    - np.einsum('...ji,...jk ->...ik', win_mu, win_mu)
inv = np.linalg.inv(win_var + (eps/win_size)*np.eye(3))

X = np.zeros((w*h, win_size, win_diam))

X = np.einsum('...ij,...jk->...ik', winI - win_mu, inv)

y = np.zeros((w*h, win_size, win_size))

y = (1 + np.einsum('aik,ajk->aij', winI - win_mu, X))/win_size
a, *_ = y.shape
y = y.reshape(a, win_diam, win_diam, win_diam, win_diam)

nz_indsCol = np.zeros(h*w*win_diam*win_diam) - 1
nz_indsRow = np.zeros(h*w*win_diam*win_diam) - 1
nz_indsVal = np.zeros(h*w*win_diam*win_diam) - 1

num = 0
print(h)

for n in range(h*w):
    for i in range(-win_rad, win_rad + 1):
        for j in range(-win_rad, win_rad + 1):
            if (w * i + j + n < h*w) and (w * i + j + n >= 0) :
                summ = 0
                for k in np.intersect1d(win_inds[w * i + j + n],
                                         win_inds[n]):
                    x1 = (n+w*i+j-k)//w
                    y1 = (n+w*i+j-k + win_rad) % w - win_rad
                    x2 = (n-k)//w
                    y2 = (n-k + win_rad) % w - win_rad
                    summ+=y[k, x1, y1, x2, y2]
                if (i == 0) and (j == 0):
                    summ = win_size - summ
                else:
                    summ = -summ

            nz_indsCol[num] = w * i + j + n
            nz_indsRow[num] = n

```

```

        nz_indsVal[num] = summ
        num+=1
    if (n%(50*w) == 0):
        print(n//(w))
L = scipy.sparse.coo_matrix((nz_indsVal[:num],
                             (nz_indsRow[:num], nz_indsCol[:num])), shape=(h*w, h*w))
return L

In [11]: def my_closed_form_matting_with_prior(L, t,
                                              lumbda = 1e-4, prior_confidence = None, consts_map=None):
    U = np.ones((t.size)).reshape(1, t.size)
    U = scipy.sparse.diags(U, [0])
    logging.info('Solving for alpha.')
    solution = scipy.sparse.linalg.spsolve(
        L + U * lumbda,
        t.ravel() * lumbda
    )
    return solution

In [12]: def get_J(A, img, f_alpha_map, t0 = 0.1):
    img = img.astype(np.int16)

    img[:, :, 0] -= A[0]
    img[:, :, 1] -= A[1]
    img[:, :, 2] -= A[2]
    z = np.maximum(f_alpha_map, t0)
    img[:, :, 0] = img[:, :, 0]/z
    img[:, :, 1] = img[:, :, 1]/z
    img[:, :, 2] = img[:, :, 2]/z

    img[:, :, 0] += A[0]
    img[:, :, 1] += A[1]
    img[:, :, 2] += A[2]

    img = np.maximum(img, 0)
    img = np.minimum(img, 255)
    return img

In [13]: def dehaze(img_, r = 15, p = 0.0001, omega = 0.95,
                  eps=1e-2, win_rad = 2, lumbda = 1e-4,
                  t0 = 0.1, filtr = False, kernel_size = 20):
    img = img_
    img = img.astype(np.int16)

```

```

rad = get_radiance(img, r = r)
A = get_atmosphere_A(img, rad, p)
alpha_map = get_t_waved(img, A, r = r, omega = omega)

L = compute_laplacian_my(img, eps = eps, win_rad = win_rad)
alpha = my_closed_form_matting_with_prior(L, alpha_map,
                                         lumbda = lumbda)
alpha = alpha.reshape((img.shape[:2]))

f_alpha_map = alpha
if (filtr):
    dst = filtering(f_alpha_map, sigma=kernel_size)
else:
    dst = f_alpha_map

f, ((ax1, ax2), (ax3, ax4)) = plt.subplots(2, 2,
                                             figsize=(15,15))

ax1.imshow(img)
ax1.set_title("Original" + str(eps))
ax1.axis('off')

ax2.imshow(get_J(A, img_, dst, t0 = t0))
ax2.set_title("rehazed")
ax2.axis('off')

ax3.imshow(rad, cmap = 'gray')
ax3.set_title("radiance")
ax3.axis('off')

ax4.imshow(dst, cmap = 'gray')
ax4.set_title("f_alpha_map")
ax4.axis('off')

return A, img_, dst, f_alpha_map, rad, L

```

```

In [14]: def filtering(image, halo=5, sigma=50):
    height = image.shape[0]
    width = image.shape[1]

    spatial_gaussian = []
    for i in range(-halo, halo+1):

```

```

        for j in range(-halo, halo+1):
            spatial_gaussian.append(
                np.exp(-0.5*(i**2+j**2)/(max(1, halo/2)**2)))

    padded = np.pad(image, halo, mode="edge")

    out_image = np.zeros(image.shape)
    weight = np.zeros(image.shape)

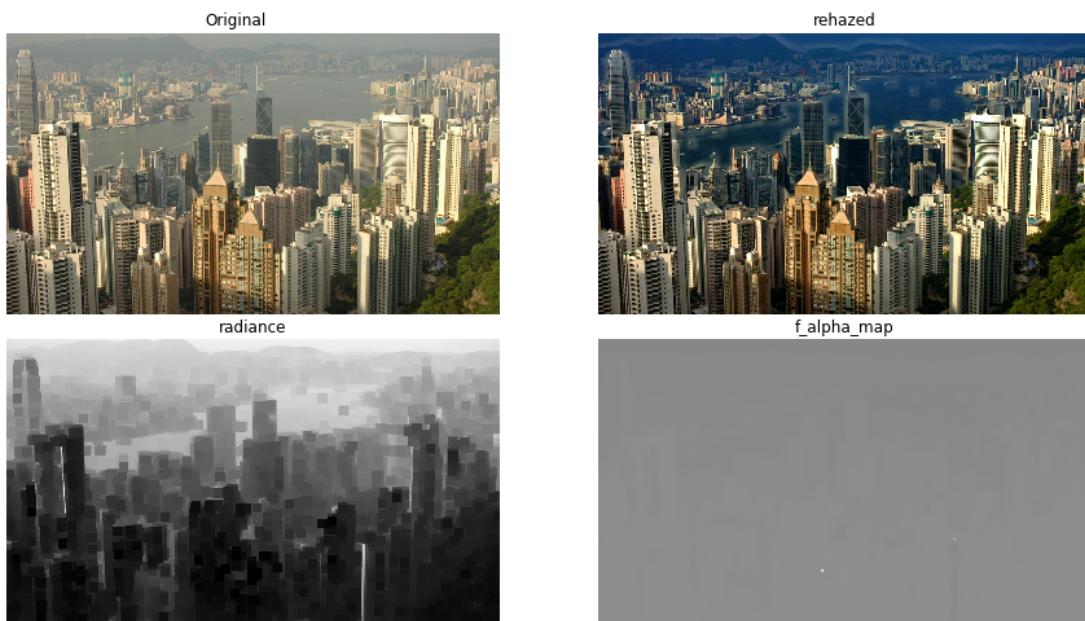
    idx=0
    for row in range(-halo, 1+halo):
        for col in range(-halo, 1+halo):
            value = np.exp(-0.5*(padded[halo+row:height+halo+row,
                                         halo+col:width+halo+col]-image)**2) \
                    *spatial_gaussian[idx]
            out_image += value*padded[halo+row:height+halo+row,
                                         halo+col:width+halo+col]
            weight += value
            idx+=1

    out_image /= weight

    return out_image

```

In [15]: A, img_, dst, f_alpha_map, rad, L = dehaze(img,
 win_rad = 2, lumbda=1e-4, kernel_size = 20, filtr = True)





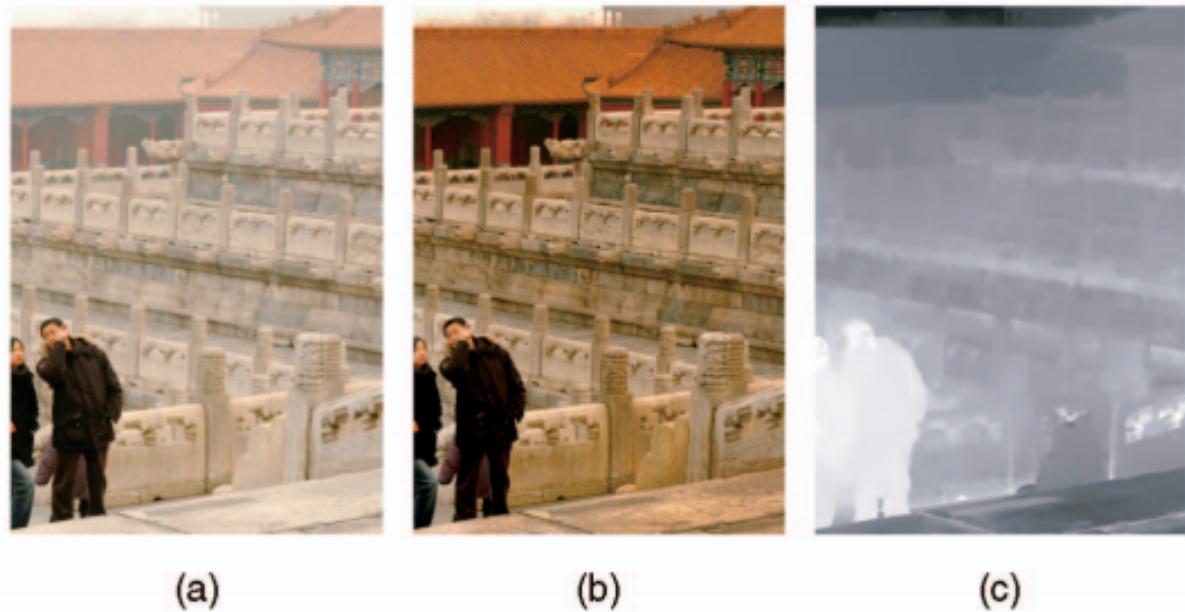


Итог:

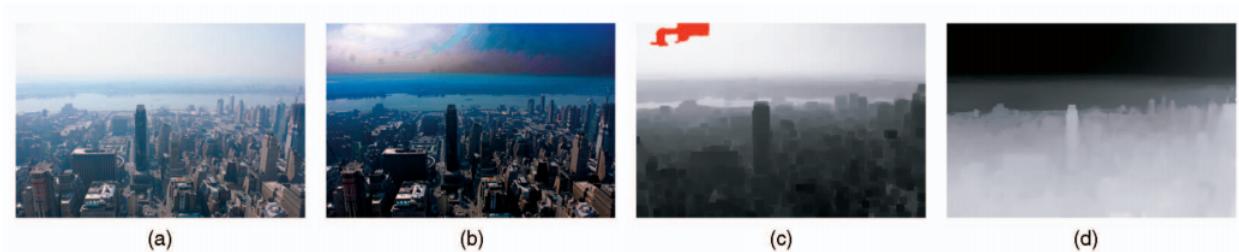
В этой статье мы разобрали очень простой, но мощный инструмент, называемый "Single

"Image Haze Removal Using Dark Channel Prior". Он основан на статистике изображений без помутнения на улице.

Так как метод основывается на статистики, он может не работать для некоторых конкретных изображений. Когда объекты сцены подобны атмосферному свету, и на них не отбрасывается тень (например, белый мрамор).



Минимальная яркость объектов имеет яркие значения вблизи таких объектов. В результате наш метод будет недооценивать передачу от этих объектов и переоценивать слой тумана. Более того, так как наш метод зависит от дымки (1), он может потерпеть неудачу, если эта модель физически недействительна. Во-первых, предположение о постоянности вклада воздуха может быть неверна. На (a), атмосферный свет яркий слева и тусклый справа. Наша автоматически оцененная А (c) не является реальной А в других регионах, поэтому восстановленная часть неба справа темнее, чем должна быть (b).



A Closed Form Solution to Natural Image Matting

С помощью анализа изображения получаем следующее предположение:

$$\mathbf{I} = \mathbf{F}\alpha + \mathbf{B}(1 - \alpha) \quad (14)$$

Отсюда, с помощью простых преобразований, мы можем получить

$$\alpha_i = \mathbf{a}^T \mathbf{I}_i + b \quad (15)$$

Функция потерь:

$$J(\{\alpha_i, \mathbf{a}_i, b_i\}) = \sum_{j=1}^N \left(\sum_{i \in w_j} (\alpha_i - (\mathbf{a}_j^T \mathbf{I}_i + b_j))^2 + \epsilon \mathbf{a}_j^2 \right) \quad (16)$$

$$\sum_{j=1}^N \left\| \begin{bmatrix} I_1^{j^T} & 1 \\ I_2^{j^T} & 1 \\ \vdots & \vdots \\ I_w^{j^T} & 1 \\ \sqrt{\epsilon} I_{3 \times 3} & 1 \end{bmatrix} \begin{bmatrix} \mathbf{a}_j \\ b_j \end{bmatrix} - \begin{bmatrix} \alpha_1^j \\ \alpha_2^j \\ \vdots \\ \alpha_w^j \\ 0 \end{bmatrix} \right\|^2 = \sum_{j=1}^N \left\| G_j \begin{bmatrix} \mathbf{a}_j \\ b_j \end{bmatrix} - \bar{\alpha}_j \right\|^2$$

$$\text{Оптимальные } \begin{bmatrix} \mathbf{a}_j \\ b_j \end{bmatrix}^* = (G_j^T G_j)^{-1} G_j^T \alpha_j$$

$$J(\alpha) = \sum_{j=1}^N \left\| G_j (G_j^T G_j)^{-1} G_j^T \alpha_j - \alpha_j \right\|^2 = \begin{bmatrix} \alpha_1 \\ \alpha_2 \\ \vdots \\ \alpha_N \end{bmatrix}^T L \begin{bmatrix} \alpha_1 \\ \alpha_2 \\ \vdots \\ \alpha_N \end{bmatrix}$$

$$\mathbf{J}(\alpha) = \alpha^T L \alpha \Rightarrow L \alpha = 0$$

Задача сводится к нахождению минимума:

$$\alpha^T L \alpha + \lambda \left(\sum_i (1 - \alpha_i) + \sum_i \alpha_i \right)^2 \rightarrow \min \quad (17)$$